

PC Support Advisor

The Essential Resource for PC Support Professionals

Update 66

How To Customize Word For Windows 6.0

You can make Word 6 work just the way your users want 3

Mike Lewis looks at some of the ways that you can speed up your users' work with the latest version of WinWord.

File: A0512.1

Understanding REXX

The key to automating OS/2 tasks 9

If you're starting to support OS/2 machines you'll need to know REXX. You may wish to teach your users the basics, too. Geoff Drake takes you through it.

File: O0513.1

Assessing PC Users' Training Needs

Before you train your PC users, it pays to ask a few questions.....17

In PCSA 64, Matthew Bell explained how to train PC users. In this article, Debi Schifreen explains in more detail the correct way to plan a training session.

File: M0209.1

Understanding PCMCIA

The key to plug-in devices for portable computers 21

Dave Stott presents the second half of his description of PCMCIA technology, a major component of modern laptop machines.

File: H0810.7

On This Month's Utility Disk

MS-DOS Hypertext guide to the Internet

Lots of sample REXX code

Latest Sound Blaster drivers for OS/2

Utility to extract all the text from a Windows .HLP file

"Double trouble" - says Robert Schifreen.

It must have been six or seven years ago that a program called Free Space, from Ashton Tate, arrived on my desk. It was a device driver for PCs, designed to be used by people who tended to keep large dBASE data files on their PCs. The driver would automatically compress these files (and others, though dBASE files compressed especially well) when dBASE wrote them to disk, thus saving lots of very expensive (in those days) space.

Free Space failed to sell in huge numbers and Ashton Tate quietly dropped it. But real-time disk compression, ie, implemented as a device driver instead of the user having to do it from a command line, was not forgotten. When Windows 3.0 resulted in a worldwide scramble for larger hard disks, a couple of enterprising software companies resurrected the idea and products like SuperStor, DoubleDisk and Stacker flourished.

Microsoft, the company whose Windows product created the need for such software, then realised that a disk compression utility would help sales of its soon-to-be-launched MS-DOS 6.0.

The initial idea was that Microsoft would bundle Stacker, but talks between the two companies collapsed.

No one seems to know the full story about why the talks collapsed, though it seems certain that money was the problem. Stac Electronics wanted lots, though Microsoft wasn't prepared to hand over such a large sum. Some observers have even suggested that Microsoft wanted Stacker for nothing, with Stac Electronics being given assurances that it would make lots of money when users wanted to upgrade their bundled copy to perhaps a more powerful "Stacker Professional" version.

With the talks having broken down, and the launch of DOS 6 getting ever closer, Microsoft decided to write its own disk compressor. It bought in technology from an external company, and modified this to integrate it tightly with the DOS 6 kernel.

From the start, DoubleSpace caused problems for Microsoft. Users around the world complained in their hundreds that they'd lost data and that they suspected DoubleSpace of

being responsible. Microsoft claimed that DoubleSpace was 100% reliable, but they brought out DOS 6.2, with an updated DoubleSpace, anyway. Microsoft was also attacked by those who disliked the way that it was trying to put companies like Stac Electronics out of business by bundling, with DOS, the product that Stac was selling as an add-on.

Stac refused to die, and decided to take legal action against Microsoft. And in March Stac won around \$120m. The jury decided that Microsoft had stolen Stac's ideas and used its patented technology. Microsoft did, though, manage to scoop back a few dollars, in its counter-suit that accused Stac of disassembling large chunks of MS-DOS in order to discover how the DoubleSpace loader worked.

Hopefully Microsoft will, in future, think more carefully before attempting to destroy any more of the comparatively small software companies which, over the years, have done so much to help sales of MS-DOS.

PCSA

PC Support Advisor

PC Support *Advisor* is a publication of **International Technology Publishing**.

ISSN: 1031-3966

Editor: Robert Schifreen

Editor in Chief: Les Bell

Publisher: Chris Carvan

Email: hex@cix.compulink.co.uk
CompuServe: 100016,301

Copyright:

All material is *Copyright* © 1994 *International Technology Publishing*, and must not be reproduced in part or full, by any means, without the written permission of the publisher.

Liability:

The information, opinions, interpretations and directions herein are based on the best information available, but their completeness and accuracy cannot be guaranteed.

Responsibility:

Responsibility for the views and opinions expressed in this publication lies with the publisher, *International Technology Publishing*, and not with any agents of the publisher nor the editors or contributing editors.

International
Technology
Publishing Inc.
North American Subscriptions
Suite 200
555 De Haro Street
San Francisco, CA94107
USA

Telephone: (415) 255 1295
Facsimile: (415) 255 8496

International
Technology
Publishing Inc.
European Subscriptions
13 Callcott Street
London
W8 7SU
United Kingdom

Telephone: (071) 724 9306
Facsimile: (071) 706 3296

International
Technology
Publishing Pty Ltd.
SE Asian Subscriptions
138 Cecil Street
Cecil Court #13-03
Singapore
0106

Telephone: (65) 227 4187
Facsimile: (65) 227 2885

International
Technology
Publishing Pty Ltd.
Australasian Subscriptions
Suite 511
185 Elizabeth Street
Sydney, NSW, 2000
Australia

Telephone: (02) 261 4683
Facsimile: (02) 261 4741

How To Customize Word For Windows 6.0

Mike Lewis looks at some of the ways that you can speed up your users' work with the latest version of WinWord.

This article describes some of the many ways in which you can customise Word 6.0 for Windows. By using the techniques which I'll describe, you will be able to automate many routine word processing tasks, help your users give their documents a consistent look, hide the more complex Word functions from people who don't need them, and quite a lot more.

But that's not all. As you explore Word, you'll realise that this is far more than an ordinary word processor. It's nothing less than a complete application development system. By using the WordBasic programming language, you will be able to create turnkey applications which go well beyond what you would normally expect from a word processor.

For example, suppose you support a group of Word users whose job is to produce invoices. You can supply them with a template containing a standard form which will guide them through the steps in the invoicing process. You can add AutoText (boilerplate text) for entering the names of products, macros for calculating taxes and discounts, and a menu for tying the whole thing together. Your template will be a complete invoicing application.

Templates

Templates are the key to customising Word. You can think of them as containers for all the Word customisation elements. They hold pre-defined styles, AutoText entries, custom menus, recorded macros, WordBasic code, and several other items. By switching between templates, you can completely change the personality of Word.

There are two types of template.

The Normal template contains global settings which are in force the whole time. You typically use it to define default typefaces and page layouts for use throughout the company. You don't have to do anything special to invoke the Normal template; it is available throughout the session.

The other type of template contains settings for a specific type of document. Users must explicitly "attach" this template to a document. To produce an invoice, for example, you would open a new document and attach the invoice template to it. The standard invoice form will then appear in the editing window, and the invoice-related menus and macros will become available. This does not replace the Normal template, which remains in force throughout.

The Word package includes around 20 pre-written templates, for common document types such as memos, fax headers and reports. It's unlikely that you'll want to keep them, but they do provide a useful starting point for designing your own templates.

Changing Templates

The easiest way to create a template is to create an ordinary document and to save it as a template. To create your invoice template, for example, you would type your standard invoice form into a new document, then save it by specifying "Document template" in the "Save File as Type" field in the Save As dialog. By default, template files have the extension DOT.

You can then continue to add text, styles, macros and other elements to the template. These are created in the normal way and are automatically saved when you save the template. To recall the template for further editing,

specify "Document template" in the "List Files of Type" control when you open the file.

You can also modify a template while you are editing the document to which it is attached. If you are editing an invoice and you decide to add a new AutoText entry, create the entry in the usual way, but select the invoice template in the "Make AutoText Entry Available To" field in the AutoText dialog. Similar methods are used when creating styles, macros etc.

Attaching A Template

You normally attach a template when you create the document, by selecting its name from the File New dialog. The template will remain attached for as long as the document exists, or until you explicitly attach a different one.

To attach a different template, open the Templates and Add-ins dialog from the File menu, press the Attach button, and select the required template from the resulting list. If you want to remove the current template without attaching a different one, select the Normal template.

Managing Template Entries

Word's Organizer dialog can help manage template entries. You use it to copy entries from one template to another, and to delete and rename entries. To access it, press the Organizer button in the Templates and Add-ins dialog.

You can also "borrow" items from other templates. For example, you might want to run one of your invoicing macros while you are editing a letter. Rather than switch templates, you can make the invoicing template available globally, on a temporary

basis. To do so, press the Add button in the Templates and Add-ins dialog, and select the required template. The template will be available for the rest of the session. To save memory, switch it off (by clearing its checkbox) when you have finished with it.

User Interface

There are three aspects of the Word user interface which you can customise: the pull-down menus, the toolbars, and the shortcut keys. In each case, the changes are stored in a template - either the Normal template or the template attached to a document.

The procedure is similar in all three cases. You call up the Customize dialog from the Tools menu, select the appropriate tab, and specify the template in which you wish to save the changes.

Menus

Word contains hundreds of built-in commands - far more than are normally visible on the pull-down menus. You can add any of these commands to any of the menus. You can also add macros, fonts, AutoText and styles to menus.

For example, you might want to add a command for inserting a section break. To do so, first locate the INSERTSECTIONBREAK command within the Commands list in the Keyboard section of the Customize dialog. For convenience, these commands are grouped in categories, which roughly correspond to the names of the built-in

menus. So to view all the commands connected with inserting, you would choose Insert from the Categories list.

Next, specify the menu in which you wish to place the command, the command's position within the menu, and the name of the command as it is to appear in the menu. Press the Add button to implement the change.

Using the Customize dialog, you can also rename and delete menu items, and add, delete and rename entire menus. The method of doing so should be fairly obvious; if in doubt, press the Help button. If you make a mess of the menus, you can restore the original settings by pressing Reset All.

Toolbars

You customise toolbars in much the same way as menus. Suppose you want to create a button for the double-underline command, which is useful if you're writing Windows Help (.HLP) files. To do so, first make sure that the target toolbar is visible. From the Toolbars tab in the Customize dialog, select the Format category and look for the button for double underlining. If you don't recognise it, click each button in turn and read its description. Once you've found it, drag it to a suitable point on the toolbar.

If you can't find a button for the required command, scroll down the Categories list until you see All Commands (or Macros, Fonts, AutoText or Styles). Select the item's name, and drag it to the toolbar. The Custom Button dialog will now appear. Use it to choose one of the general-purpose

buttons. If none of them takes your fancy, choose the text button, and type a caption for it in the "Text Button Name" field. Alternatively, press Edit to design your own button.

Shortcut Keys

Using the Keyboard tab in the Customize dialog, you can assign shortcut keys to any command, macro, font, AutoText entry or style. You can also assign shortcuts to special symbols like the em dash and the copyright sign. A shortcut is any key, modified by any combination of Ctrl, Shift and Alt. It may optionally be followed by a second key, optionally modified by Shift.

To assign a shortcut, first locate the command (or macro, font etc) in the Customize dialog, just as you did when customising the menus and toolbar. If there is already a shortcut assigned to the command, this will be displayed in the Current Keys list. Press your chosen key combination in the "Press New Shortcut Keys" control. To complete the process, press the Assign button.

Macro Recording

Like many Windows applications, Word has the ability to record macros. This involves recording the steps needed to accomplish a commonly performed task, then playing them back with a single keystroke or mouse action. For example, you might record a macro which saves the current document, selects a printer, and prints the document. By assigning the macro to a menu command, toolbar button or shortcut key, you reduce the whole job to a single step.

Unlike other applications, Word does not record the actual keystrokes or mouse movements used to accomplish the task. Rather, it records the internal commands which produce the desired result. For example, if you record a macro which switches to page layout view, Word records the VIEW-PAGE command, not the keystrokes which open the View menu and select the Page Layout option. This means that your macros will continue to work even if Microsoft (or you) change the layout of the menus.

```
Sub MAIN
ToolsOptionsView .DraftFont = 1, .WrapToWindow = 0,
.PicturePlaceHolders = 0, .FieldCodes = 0,
.BookMarks = 0,
.FieldShading = 1, .StatusBar = 1, .HScroll = 1,
.VScroll = 1,
.StyleAreaWidth = "0 cm", .Tabs = 0, .Spaces = 0,
.Paras = 0,
.Hyphens = 0, .Hidden = 0, .ShowAll = 0,
.Drawings = 0,
.Anchors = 0, .TextBoundaries = 0, .VRuler = 0
End Sub
```

Figure 1 - The ToggleDraft macro, as recorded

Word For Windows 6

You won't be surprised to know that macros are stored in templates. When you record a macro, you choose to place it in either the Normal template or the template attached to the current document. Going back to our invoicing example, you might record a macro which applies discounts to an invoice. By saving it in the invoice template, the macro becomes available whenever the user is working on an invoice - and at no other times.

Recording A Macro

To start recording, either double click on "REC" in the status bar, or select Tools Macros and press the Record button. You will see a dialog in which you specify a name for the macro, the template in which you wish to store it, and an optional description to remind you what the macro does.

When you press the OK button in this dialog, a tiny recorder icon attached itself to the mouse pointer. This indicates that recording is in progress.

Also, the recorder toolbar appears. This has two buttons, to stop recording and to pause recording respectively.

You can now go ahead and record the macro. When you have finished, either press the Stop button in the recorder toolbar, or double-click on "REC" again.

Playing It Back

One way to play back your macro is to open the Macros dialog (from the Tools menu), select the required macro, and press Run. If you don't see the macro name displayed, check that the correct template is specified in the "Macros Available In" control. But this is not a particularly convenient method because it requires several keystrokes, thus partly defeating the object. Also, there is a risk of the user accidentally altering the macro by pressing the Edit button.

A better solution is to assign the macro to an interface element, that is, a menu, toolbar button or shortcut key

- or perhaps all three. You can do this before you start recording, by pressing the appropriate button in the Record Macro dialog. This will invoke the Customize dialog, with the corresponding tab selected. You then create the interface element in the usual way. When you close the dialog, you go straight into recording mode.

Alternatively, you can assign the macro retrospectively. To do so, open the Customize dialog from the Tools menu, select the required tab, and scroll down the list of categories until you see Macros. Then proceed in the usual way.

If you entered a description of the macro in either the Macro or the Record Macro dialog, it will be displayed on the status bar when the user selects the appropriate menu item or toolbar button. This helps the user understand what the macro does. However, there is no way of adding ToolTips to a button (ToolTips is the short description which pops up when the mouse pointer rests on a button for half a second or so).

```
Sub MAIN
  ToolsOptionsView.DraftFont = 1
End Sub
```

Figure 2 - Simplified ToggleDraft macro; only switches to draft font.

```
Sub MAIN
  Dim TovRec As ToolsOptionsView
  GetCurValues TovRec
  If TovRec.DraftFont = 1 Then
    TovRec.DraftFont = 0
  Else
    TovRec.DraftFont = 1
  End If
  ToolsOptionsView TovRec
End Sub
```

Figure 3 - Improved ToggleDraft macro; toggles draft font on and off

```
Sub MAIN
  Dim CountRec As ToolsWordCount
  GetCurValues CountRec
  Insert "Number of words: " + CountRec.Words
End Sub
```

Figure 4 - Macro to insert a word count

WordBasic

So far, the customisation features which we have explored do not require any form of programming and can be undertaken by knowledgeable end users. The remainder of this article deals with a more advanced topic, the WordBasic programming language.

WordBasic is a full programming language, similar to Microsoft's Visual Basic, but without VB's object-oriented extensions. It has been specially designed to interface with the Word environment, for example by allowing the programmer to execute Word commands and to work directly with the text in a Word document.

You can use WordBasic to extend and customise Word in ways that would not otherwise be possible. You can also use it to create complete applications. For example, you could create an application which prompts the user for information relating to a customer's order, then generates an order form, invoice and delivery note automatically. You can even use the language to communicate with other

applications, for example to obtain a customer's address from an Access database.

WordBasic is a big language, and I can only give you a quick introduction to it here. If you are already familiar with Basic-like languages, refer to the text box for a summary of WordBasic syntax.

Learning WordBasic

If you want to do any serious Word-Basic programming, you should

obtain the Microsoft Word Developer's Kit. It is published by Microsoft Press, at around \$40 (ISBN 1-55615-630-8). It consists of the full language manual, along with a disk containing sample code and other useful items.

However, you do not have to buy the Developer's Kit in order to get started with WordBasic. The language is an integral part of Word, and can be used without any additional software. Word's help system includes the language reference, which you access by

clicking on "Programming with Microsoft Word" in the contents page.

WordBasic And Macros

WordBasic is closely associated with Word macros. When you record a macro, Word generates WordBasic code, which you are free to inspect and edit. A macro is the basic unit of Word-Basic - similar to procedures and functions in other languages.

The easiest way of creating Word-Basic applications is often to generate as much code as possible using the macro recorder, and then to refine it by hand editing. Let's walk through an example of that.

Suppose you want a quick way of toggling the edit window between draft and normal fonts. In Word 2.0, you could do that with a single menu option, but Word 6.0 requires you to descend into the Options dialog, select a tab, and click on a checkbox. We will create a macro that does the job in one step.

Start by switching on the macro recorder. Name the macro, say, ToggleDraft. With recording in progress, select Tools Options, select the View tab, click on the Draft Font option, and close the dialog. Stop recording. Finally, use the Customize dialog to add the macro to the View menu (or to a toolbar if you prefer).

You can now test your macro by selecting the new menu item. If all is well, the editing window should switch to the draft font.

There are, however, a couple of problems. First, the macro is supposed to toggle the display, but selecting the menu item a second time does not make the normal fonts re-appear. To demonstrate the other problem, use the "Show/hide non-printing characters" button on the standard toolbar to display paragraph marks. Now, when you select the ToggleDraft menu item, the paragraph marks will disappear, which is not what you wanted the macro to do.

To understand these problems, recall what I said earlier: A macro records commands, not keypresses or mouse clicks. In this case, the "command" is the Customize dialog, with various options set. The fact that you

```
Begin Dialog UserDialog 464, 194, "Collection Details"
  Text 16, 30, 57, 13, "&Amount"
  TextBox 16, 45, 145, 18, .Amount$
  Text 16, 80, 71, 13, "&Date due"
  TextBox 16, 95, 145, 18, .DateDue$
  GroupBox 243, 30, 204, 84, "Paragraph to insert"
  OptionGroup .StPara
    OptionButton 253, 45, 180, 16, "&Suspend
                                creditnow", .SuspNow
    OptionButton 253, 65, 180, 16, "Suspend in
                                &7days", .Susp7
    OptionButton 253, 85, 180, 16, "&Legal action
                                threat", .Legal
  OKButton 210, 148, 88, 21
  CancelButton 326, 149, 88, 21
End Dialog
```

Figure 5 - Code to display the Collection Details dialog

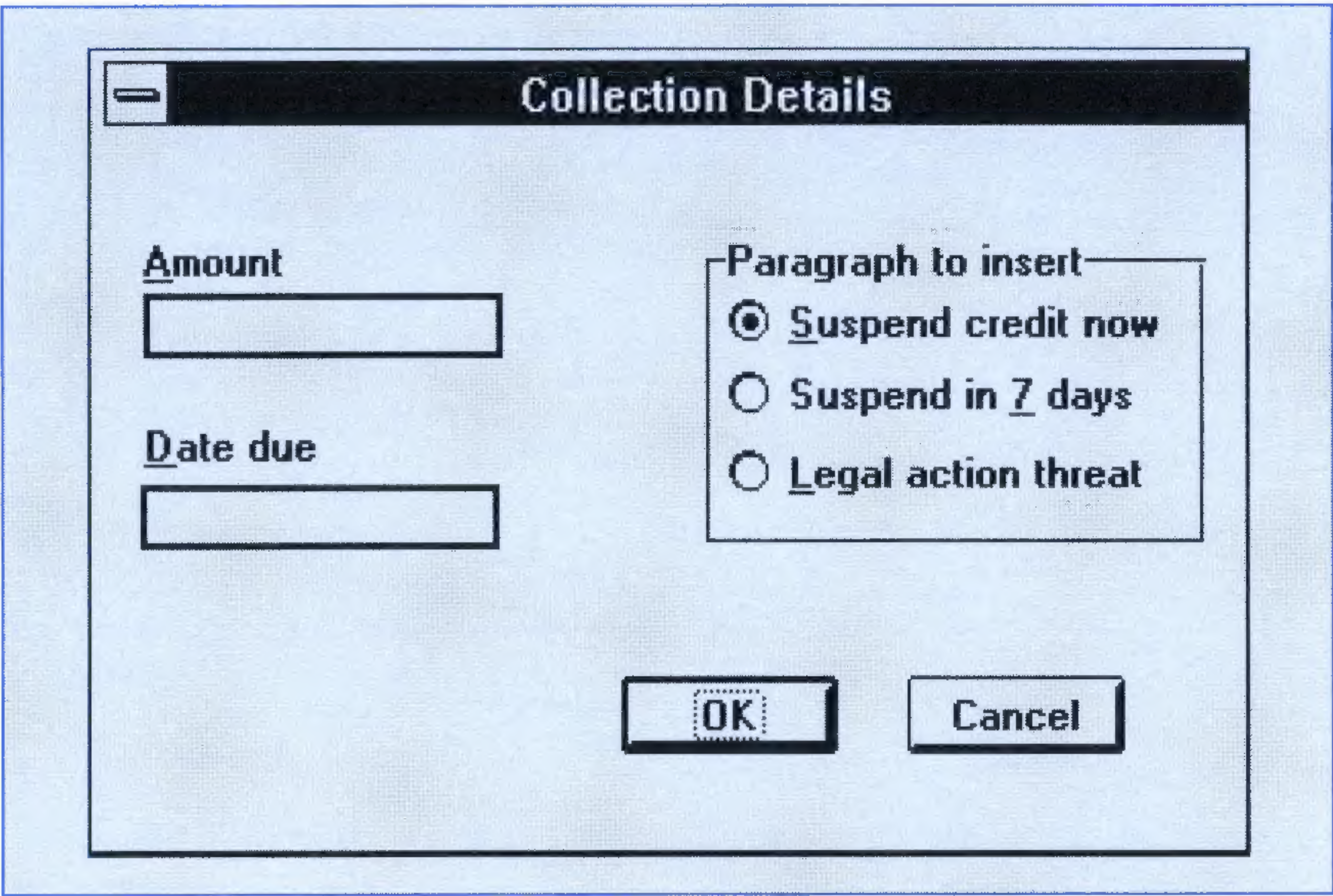


Figure 6 - The dialog created by the code in Figure 5

Word For Windows 6

clicked on the Draft Font option is irrelevant - the macro will always set the options to the way they ended up when you finished recording. To correct these problems, we need to edit the macro.

Macro Editing

To edit a macro, highlight its name in the Macro dialog and press Edit. An editing window will open, with the macro's WordBasic code displayed. This window is a cut-down version of the normal Word editing window. It includes a useful macro toolbar, which contains various buttons for recording, running and debugging macros.

When you open the editing window for the ToggleDraft macro, you will see the WordBasic code shown in Figure 1. The macro consists of a single statement, sandwiched between "Sub MAIN" and "End Sub" delimiters.

The statement is the ToolsOptions-View statement, which corresponds to the View tab in the Options dialog. Each of the statement's arguments (the words starting with a full stop) corresponds to one of the options in the dialog.

The value of the argument indicates the state of the option. For example, Paras is assigned a value of zero, which means that the macro will switch off the option which displays paragraph marks.

To prevent the macro from setting unwanted options, delete the corresponding arguments - which, in this case, means everything except the DraftFont argument. You should now have a macro like the one in Figure 2, which sets the Draft Font option and nothing else.

Dialog Records

Macros often need to test the settings within a dialog, and to take different actions according to their values. To do this, you create a special variable called a dialog record. You use a Dim statement to create the variable, and a GetCurValues statement to assign values to it. Once the values have been assigned, you can manipulate them just like any other variables.

Figure 3 shows an improved version of the ToggleDraft macro. It starts by creating a dialog record called TovRec. It then executes a GetCurValues statement, which places the current settings from the dialog into the record. Next, the macro tests the Draft Font field, and reverses its current setting. Changing a field within a dialog record does not alter the setting within Word, so the final action of the macro is to pass the setting back to the dialog. This version of the macro therefore acts as a true toggle.

For another example, take a look at the macro in Figure 4. This uses the ToolsWordCount dialog to insert a word count into the current document.

Communicating With Users

WordBasic offers several methods of communicating with the user. You can display messages and obtain responses via message boxes, input boxes and the status line. The following statement, for example, displays a message box:

```
reply = MsgBox("Hello World", "Hi There", 292)
```

The "292" argument says that the message box should include a question mark icon and two buttons, labelled Yes and No. The reply is -1 if the user chooses Yes, 0 otherwise.

In the next example, the user is invited to enter text into an input box:

```
NewMess$ = Input$( "Enter message", "Title", DefMess$)
```

DefMess\$ contains the default text displayed in the box; NewMess\$ contains the new text entered by the user. Title is the title of the window.

```
If CDRec.StPara = 0 Then
    InsertFile .Name = "SUSPNOW.DOC"
ElseIf CDRec.StPara = 1 Then
    InsertFile .Name = "SUSP7.DOC"
Else
    InsertFile .Name = "LEGAL.DOC"
End If
```

Figure 7 - Code to decide which paragraph to insert

Custom Dialogs

WordBasic also allows you to design your own dialogs, complete with list boxes, check boxes, radio buttons, and other Windows controls.

Figure 5 shows an example of this in action, and Figure 6 shows the results. The syntax is fairly complex, involving a separate sub-statement for each of the dialog's controls. The numeric values in the statement indicate the coordinates, heights and widths of the various elements.

Fortunately, you don't have to code these statements by hand. Word has an interactive dialog editor, which you use to create dialogs visually on the screen. The dialog editor is a separate program, with its own Program Manager icon. It lacks a help file, but the mechanics are reasonably obvious. The Microsoft Word Developer's Kit contains detailed instructions for its use.

Once you have created your dialog, you use the Windows clipboard to transfer it to a macro editing window. This converts the visual dialog into WordBasic code. You can also do the transfer in the other direction, in which case the code is converted back into a dialog which you can edit.

Having created the dialog, you use the Dim statement to assign a dialog record to it:

```
Dim CDRec as UserDialog
```

and a Dialog() function to make it visible:

```
reply = Dialog(CDRec)
```

You can then access the individual controls within the dialog. The example in Figure 7 shows how to use the radio buttons to decide which standard paragraph to insert.

Going Further

To finish this article, I'll briefly introduce a couple of the more advanced features of Word. You can explore these, and other advanced topics, by referring to the Microsoft Word Developer's Kit.

DDE And OLE Automation

Word supports dynamic data exchange (DDE), both as a client and server. It can request information from other applications, respond to other applications' requests for information, and execute other applications' commands. For example, you could write a WordBasic macro which opens an Excel worksheet, sends it a set of figures, and instructs Excel to perform calculations on them. To find out more, study the six WordBasic statements whose names begin with "DDE".

Although DDE is well established, Microsoft says that the preferred method of inter-program communication is now OLE Automation. This is a fairly new technique, which is based on version 2 of the Object Linking and

Embedding (OLE) protocol. In OLE Automation, an application provides "objects" for use by "container" applications. Word provides a single object, called "Basic". By using this object, other OLE Automation-compliant applications, such as Excel 5.0 and Visual Basic 3.0, can invoke WordBasic statements and functions just as if they were part of their own languages.

As its name suggests, Word's OLE Automation object is limited to the WordBasic language. It cannot be used to give container applications direct access to Word documents. Instead, the container application issues WordBasic commands and functions which act on the document. Also, Word has no way of acting as a container itself, that is, it cannot access OLE Automation objects in other applications.

Custom Wizards

A wizard is a series of dialogs which guide the user through a complex task. They are appearing in most Microsoft applications. In Word 6.0, wizards are used to help the user create certain types of document.

Using WordBasic, it is possible to

create your own wizards. In fact, a Word wizard is nothing more or less than an ordinary WordBasic macro, albeit a fairly complex one. To be consistent with the Microsoft-supplied wizards, however, it should conform to certain design and interface conventions.

The Microsoft Word Developer's Kit contains two special wizards, designed specifically to help you create your own wizards. These create a skeleton wizard, containing the common interface elements, such as the Next, Previous and Finish buttons. You then add further WordBasic code to perform the particular task that you want your wizard to achieve.

Wizards are stored as template files, but with the extension WIZ rather than the usual DOT. Word displays the wizards in the template list in the File New dialog, adding the word "Wizard" to their file names. So if you create a wizard called INVOICE.WIZ, it will appear in the File New dialog as "Invoice Wizard".

In Conclusion

Word 6.0 for Windows is a vast application, offering many opportunities for customisation and automation. This article has introduced the main topics, but there's plenty more waiting to be investigated. If you support Word users, or you are interested in developing Word applications, you'll want to spend more time exploring this fascinating program.

PCSA

An Overview Of WordBasic Syntax

- A macro begins with "Sub MAIN" and ends with "End Sub".
- A statement is terminated by a hard carriage-return, with long statements being wordwrapped by the editor. To force a line break within a statement, end the line with a backslash.
- WordBasic code is not case-sensitive.
- A comment is a line beginning with "REM", or any text beginning with an apostrophe.
- Variable names begin with a letter, contain only letters, digits, and underscores, and are subject to a maximum of 40 characters. Reserved words cannot be used as variables. Variables may be either string or numeric. String variables end with a dollar sign. Numeric variables have no special ending.
- WordBasic supports these arithmetic operators: +, -, *, / and MOD; these relational operators: =, <>, >, <, >= and <=; and these logical operators: AND, OR and NOT. In a condition, a value of -1 is true, 0 false.
- WordBasic supports the following control structures: If/Then/Else/End If; While/Wend; For/Next; and Select Case/Case/End Select.

The Author

Mike Lewis is a freelance PC, Database and Windows consultant.

Understanding REXX

If you're starting to support OS/2 machines you'll need to know REXX. You may wish to teach your users the basics, too. Geoff Drake takes you through it.

To the PC support person, MS-DOS batch files are wonderful things. You can customise the look and feel of a PC with just a few carefully-crafted batch files, and you can use them to automate routine tasks that you or your users have to perform.

In the world of mainframes and minis, batch programming languages (or JCLs, for Job Control Languages, as they're often called) are far more sophisticated. Shell scripts under UNIX, for example, can do just about anything that you or your users could ever want.

REXX, the REstructured eXtended eXecutor, made its first appearance on IBM mainframes, developed by Mike Colinslaw of IBM UK. Mike developed REXX as a replacement for IBM's equivalent of the DOS Batch language. This was distributed internally and soon proved to be popular within IBM. Over the years it has been adopted by many people, and there are now versions for DOS, OS/2 (where it is supplied as standard), the Commodore Amiga, Apple Mac and others.

What makes REXX popular is its simplicity. The character string is its only data type, for example. Variables do not have to be defined before you use them. It has a powerful facility called stem variables, of which more later. It has all the modern constructs for structured programming. And finally, it is easily extendible, allowing it to interface to the resident operating system and any sub systems.

Getting Started

All the examples in this article were developed under OS/2 V2.1. Under OS/2, REXX is integrated with the operating system. You use your favourite editor to write a file (with the suffix .CMD).

When OS/2 executes this file it checks the first line. If it is a comment,

OS/2 calls the REXX processor to execute the file, otherwise the OS assumes it's looking at an OS/2 batch file.

No true programmer can introduce a language without using the "Hello World" example introduced by Kernighan and Richie, so Figure 1 does just that.

```
/* Hello World in REXX */
SAY "Hello World"
EXIT
```

Figure 1 - Hello World

```
/* Using a Variable and reading User input */
SAY "Please enter your first name"
PULL userName
SAY "Hello to you" userName
EXIT
```

Figure 2 - The use of Variables

```
/* Calculating File sizes */
file1 = "C:\COMMAND.COM"
file2 = "C:\AUTOEXEC.BAT"
file3 = "C:\DOS5\COUNTRY.SYS"
size1 = STREAM(file1,'c','query size')
size2 = STREAM(file2,'c','query size')
size3 = STREAM(file3,'c','query size')
totalSize = file1 + file2 + file3
avgSize = totalSize / 3
SAY "Total Size = " totalSize ", Average File Size = " avgSize
EXIT
```

Figure 3 - Calculating File sizes.

```
/* Will these files fit on a Floppy */

file1 = "C:\COMMAND.COM"
file2 = "C:\AUTOEXEC.BAT"
file3 = "C:\DOS5\COUNTRY.SYS"
size1 = STREAM(file1,'c','query size')
size2 = STREAM(file2,'c','query size')
size3 = STREAM(file3,'c','query size')
totalSize = file1 + file2 + file3
floppySize = 1.44 * 1024 * 1024
IF totalSize < floppySize
    THEN SAY "These files will fit on a 1.44MB Floppy"
SAY "Total Size = " totalSize
SAY "Floppy Size = " floppySize
EXIT
```

Figure 4 - Will these files fit on a Floppy?

```
.....
IF totalSize < floppySize
    THEN SAY "These files will fit on a 1.44MB Floppy"
ELSE
    SAY "These files are too big for a single Floppy"
.....
```

Figure 5 - The ELSE construct

Although this example looks fairly simple, it does illustrate some major concepts. First, a REXX program must start with a comment, for reasons explained above. Comments in REXX start with /* and end in */, just like C. You'd normally use this first line to state the purpose of the program, perhaps the date it was created, and some history.

Next we have the command for writing a message to the standard output device, SAY. We have also introduced the concept of a character string. This is a sequence of characters that make up a message, or name etc. In REXX you can use a single quote, or a double quote pair to delimit the string.

The advantage of having two methods of defining a string is that you can use the alternative pair if the string contains the other character, eg, "It's a cruel world".

To finish a REXX program you use the EXIT keyword. This is not strictly necessary, but it is a good habit to use it. We will see why later.

Variables

Variables in REXX are all treated as character strings. Character strings that contain numbers may be used in arithmetic operations (added to, subtracted from etc), while strings that hold characters may not. Figure 2 shows our second example. This time we output a message to the user and then read the value they type in. Here we have two new concepts. We need an area to hold the user's name. We call this area UserName. We do not need to know the contents of the area, we simply refer to the name we have given it.

The second concept is reading input from the user. REXX does this by using the PULL keyword. This means fetch one word from the standard input device, in this case the keyboard.

Variables can also be used in calculations.

Figure 3 shows a simple program for calculating the total size of a number of files. Here we introduce the concept of a function.

Keyword commands tell REXX to perform some action, SAY, PULL etc.

On the other hand, a function is given a number of values. It uses these values to perform a calculation and then returns the result. REXX has many built-in functions. Some of these are part of the language, some are extensions to the language provided by a particular implementation. The one we use in this example is STREAM. STREAM is used with file input and output, and this will be covered later. For now all we need to know is that calling STREAM with these parameters will return the current file size.

The example has three variables, file1, file2 and file3. Each of these holds the name of a file. We assign a new value to a variable by using the assignment operator =. Next, STREAM is called three times, each time we pass the appropriate filename (using the variable), and assign the returned file size to a new variable.

Our three size variables now contain the sizes of our files. The next step is to perform two calculations. The first adds the three file variables together and assigns the result to totalSize. The second calculation produces the average by dividing totalSize by 3.

The last step of the program is to print the result. Here we find a new feature of SAY. If we place several strings, or variables, on a SAY, the message is printed on a single line.

Decisions

So far we have seen how to display messages and results, and how to get input from the user and perform calculations. We often want to perform different actions based on our data. In Figure 3, we may be trying to calculate whether or not our files will fit on a single floppy disk. REXX has the IF keyword to provide the ability to take different paths based on the result of a decision.

Figure 4 shows Figure 3 modified to make a decision. Here we output a message if the files will fit on a 1.44 MB floppy. The IF construct is given a test value, in this case totalSize < floppySize. If the result is true, the statement after the THEN clause is executed. If the test is false, the statement is skipped.

REXX has an ELSE clause, too. Figure 5 shows the IF statement in Figure 4 rewritten to use an ELSE construct.

```
IF totalSize < floppySize THEN
DO
    SAY "These files will fit on a 1.44MB Floppy"
    /* Code to copy File 1 */
    /* Code to copy File 2 */
    /* Code to copy File 3 */
END
ELSE
    SAY "These files are too big for a single Floppy"
    . . . . .
```

Figure 6 - The DO/END construct

```
/* */
x = "C:\COMMAND.COM"; y = "C:\AUTOEXEC.BAT"; z =
    "C:\DOS5\COUNTRY.SYS"
a = STREAM(file1,'c','query size');size2 =
    STREAM(file2,'c','query size');size3 =
    STREAM(file3,'c','query size')
t = x+y+z; f= 1.44 * 1024 * 1024; IF totalSize < floppySize
THEN SAY "These files will fit on a 1.44MB Floppy";
/* Code to copy File 1 */ /* Code to copy File 2 */
/* Code to copy File 3 */ ELSE
SAY "These files are too big for a single Floppy"
SAY "These files will fit on a 1.44MB Floppy"
SAY "Total Size = " t ;SAY "Floppy Size = " f
EXIT
```

Figure 7 - Who coded that then?

Understanding REXX

Groups of Statements

Often, as the result of a decision, we want to execute more than one statement. The IF/THEN/ELSE construct only executes the statement immediately after the THEN or ELSE. If we need to execute more than one statement we must enclose them in a DO/END group. Figure 6 shows a DO/END group used to copy the files. Copying files is possible from within REXX - we will see how later.

At this point it might be a good idea to emphasise a few points on style. There are some approaches you can take when writing your program that will aid you in the future, and will help anyone else who needs to understand your code.

Use comments liberally. What was blindingly obvious when you wrote the program, may be rather more obscure a year later. Make your variable names meaningful. It is easier to get some idea of a variables function if it is called FileCount than if it is called xyzzy. Use indentation to show the structure of control flow - REXX doesn't mind about leading spaces. Indenting code at loops, IFs and DO/END blocks helps you visualise what is happening. If you do this, you must be consistent and diligent - forgetting to remove the indentation when you change the code can cause more trouble than its worth.

Figures 7 and 8 show two different versions of the same program, to illustrate how clear coding style improves the readability of the program. (Incidentally, there's a coding error in these programs. Try asking your users to debug both of them, and see which one they prefer working on.)

Loops

The remaining control construct we need to cover is the loop. A loop allows us to execute the same section of program a number of times. There are a multitude of reason why we may need a loop. We might want to keep a total count. We may need to execute our code against a number of different objects and so on.

There are several types of loop construct. These differ in when the control

```

/* This program copies a number of files onto a floppy.
   Before it executes the copy it checks that the files
   will fit onto a 1.44MB diskette.
   Written by: Fred Smith
   Version: 1.0 2 Feb 1994 FS Initial Version
           1.1 9 Feb 1994 FS Corrected size
           calculation to allow for directories. */
/* Set up a number of variables to hold the names
   and paths of the files to be copied.
   These should be full path names because we cannot
   guarantee the directory from which it is run */
file1 = "C:\COMMAND.COM"
file2 = "C:\AUTOEXEC.BAT"
file3 = "C:\DOS5\COUNTRY.SYS"
/* Use the stream command to get the actual size
   (in bytes) of each file */
size1 = STREAM(file1,'c','query size')
size2 = STREAM(file2,'c','query size')
size3 = STREAM(file3,'c','query size')
/* Add the sizes together to give the total byte count */
totalSize = file1 + file2 + file3 /*
   Calculate the size of a floppy (1MB = 1024KB =
   1024*1024) take 1024 bytes off to allow for the
   directory and boot sectors etc. */
floppySize = 1.44 * 1024 * 1024 - 1024
/*
   Check size and if OK copy the files
   Otherwise write an error message
*/
IF totalSize < floppySize
    SAY "These files will fit on a 1.44MB Floppy"
    /* Code to copy File 1 */
    /* Code to copy File 2 */
    /* Code to copy File 3 */
ELSE
    SAY "These files are too big for a single Floppy"
/* Output the size values */
SAY "Total Size = " totalSize "
SAY "Floppy Size = " floppySize
/* Program complete, exit to caller */
EXIT

```

Figure 8 - A better example

```

/* This program accepts a command from the user. This command
   can be one of three
   options:
       T, this will type config.sys and autoexec.bat
       to the screen.
       P, this will print config.sys and autoexec.bat
       to the default printer
       Q, this will exit the program. */
Command = "?" /* Initialise the test variable */
DO WHILE Command \= "Q" /* Terminate the loop on Q */
    SAY "Enter Command (T,C,Q)" /* Output message*/
    PULL Command /* Read user command */
    IF Command = "T" THEN /* If T for Type */
        DO /* Execute type commands */
            "Type CONFIG.SYS"
            "Type AUTOEXEC.BAT"
        END
    ELSE IF command = "P" THEN /* Otherwise check for
                                Print */
        DO /* If true execute
                                print commands */
            "copy CONFIG.SYS PRN:"
            "copy AUTOEXEC.BAT PRN:"
        END
END /* Terminate the While Loop */

```

Figure 9 - A WHILE Loop

test is executed. A DO/WHILE loop will execute the test first, and then execute the statements if the test was true. A DO/UNTIL loop will execute the statements once, and then test for a true condition. These are known as entry condition and exit condition loops respectively. REXX also allows you to build "forever" loops. These are loops that will always give true on the test condition. Normally, these use some other method for terminating the loop.

Figure 9 shows a typical loop. In this loop we set up a control variable (Command) and, while this flag is not equal to "Q", we continue to execute the program. One point to remember, variables used for testing must be initialised before using them in a test.

In the tests we use upper case values, eg, "Q". By default REXX will translate all input to upper case. This is a point to remember when you are having difficulties. This translation can be avoided by using PARSE PULL instead of the normal PULL command.

There are several other loop constructs in REXX. One of these is the DO number/END construct. In this case the loop will be executed the number of times specified. Eg, DO 6 / END will execute the loop six times. There is also an incremental loop. DO x= number TO number/END. This will set the variable (x) to the first number, and then execute the loop. The number is incremented and tested against the second number. If it is below, or equal to the second number, the loop is executed again.

Calling Other Code

When designing programs it helps to break the code into manageable sections. This helps the maintainability of the code and may allow us to reuse sections of it in other programs. As an example, we could have some code that checks that there is enough room on the destination disk before it copies a file.

There may be several points in the program where this code could be used. One method we could use would be to re-enter the code at each point in our program. This is bad prac-

tice for several reasons. First, it makes our code much larger. Second, if we find a bug in the code we have to locate all the occurrences of that code and change them.

REXX has three ways of sectioning code. These are subroutines, subsystems, and external programs. REXX also has a rich set of built-in functions. These are routines that have a general use, such as finding the maximum of two numbers or, locating a string within a phrase.

Subroutines

A subroutine is a section of code that may be called from another point in your program. You may, optionally pass some parameters to it. This code is executed and a value is returned. A

subsystem is usually provided to act as an interface to an external system. For example, OS/2 REXX has several subsystems that may be registered with REXX. These allow you to access data in a SQL database, to gain access to TCP/IP sockets, or to access NetBIOS calls from within REXX.

The final option, external programs, allows you to pass unknown

```

. . .
CALL mysub
. . .
EXIT
mysub:
PROCEDURE
. . .
RETURN

```

Figure 10 - A subroutine skeleton

```

/* Program to copy two files using a subroutine */
CALL fcopy "CONFIG.SYS" "c:\xx"
CALL fcopy "AUTOEXEC.BAT" "c:\xx"
EXIT
/* This subroutine takes two parameters, a file name
   and a destination. Destination can be any construct
   that is valid for the COPY command */
fcopy:
PROCEDURE
ARG fname dest
'copy ' fname dest '/V'
/* first call fname=CONFIG.SYS, second call fname=AUTOEXEC.BAT */
/*
RETURN

```

Figure 11 - Passing Parameters

```

/* Subroutine without PROCEDURE statement */
x = 21
z = square(4)
say 'x = ' x /* prints 16, not 21 */
EXIT
square:
ARG sq
x = sq * sq
RETURN x

```

Figure 12a - Subroutines without PROCEDURE statement

```

/* Subroutine with PROCEDURE statement */
x = 21
z = square(4)
say 'x = ' x /* prints 21 */
EXIT
square:
PROCEDURE /* PROCEDURE 'protects' the new X value */
ARG sq
x = sq * sq
RETURN x

```

Figure 12b - Subroutine with PROCEDURE statement

Understanding REXX

commands to a default processor. In OS/2's case this is the standard OS/2 command processor. We have already encountered this option, calling an external program, in Figure 9. When

REXX is given a command that is not a REXX command it passes this command to the command processor, which attempts to execute the string as an OS/2 command. This gives you a

very smooth method for integrating automatic OS/2 command scripts.

Subroutines are normally defined in the same file as the main program. The code is placed after the EXIT keyword. The EXIT keyword is optional in REXX, except when you have defined one or more subroutines, otherwise the program drops through from the end of your program into the top of the subroutine.

Each routine must be identified by a unique name. This name is placed on a line, followed by a colon, eg, MYSUB:. This construct is called a label. Lines of code are placed after the label. The end of a subroutine is delineated by a RETURN keyword. Figure 10 shows the skeleton construct for a function or subroutine.

To make subroutines more general purpose, we can pass values into them. These values can be used to customise the subroutine. For example, if we wrote a routine to copy a file, we would probably want to pass two parameters - the name of the file to be copied and the new destination. This is done by using the ARG keyword. Figure 11 shows the use of the ARG keyword. Values passed to the subroutine are mapped onto the names in the ARG list. Inside the subroutine we can use FNAME and DEST. Each time the subroutine is called, FNAME and DEST contain the current values passed for each parameter.

Parameters are passed by position. If you have three values in your ARG list, then those three parameters must be passed in that order on the CALL statement. You cannot leave parameters out - they must appear. There are two things to watch out for with argument passing. First, the passed list is treated as a single string, so each word is treated as a single argument. Second, the string is translated into upper case.

There are techniques to overcome these problems. Normally the names on an ARG statement are separated by blanks - if you separate them by commas each value is treated as the complete string between the commas, and, instead of using the ARG command you can use PARSE ARG. This does not translate the arguments to upper case.

```
/* Calculate the average of a set of values
   (using stem arrays) */
result.0 = 5      /* set up array */
result.1 = 10.5
result.2 = 11.2
result.3 = 9.7
result.4 = 8.6
result.5 = 12.1
total = 0        /* Set base total to zero */
DO x = 1 TO result.0
total = total + result.x
END
say "Average = " total /result.0
EXIT
```

Figure 13 - Using STEM Variables as Arrays

```
/* Example of using Stem variables as structures*/
/* First Setup User Fred's Data */
User1.Name = 'Fred'
User1.Password = 'xyzzy'
User1.ConnectDate = '01-02-1994'
User1.MachineType = 'IBM PS/2 95'
/* Setup User Janice's Data */
User2.Name = 'Janice'
User2.Password = 'zyxxz'
User2.ConnectDate = '02-02-1994'
User2.MachineType = 'IBM PS/2 295'
/* Print the data */
call PrintName 'USER1'
say ' '
call PrintName 'USER2'
EXIT
/*
   This routine is passed the name of a stem variable that
   holds the data to be printed.
*/
PrintName:
ARG ID      /* ID = Stem of User to Print */
/*
   REXX cannot use a passed STEM name directly, so we have
   a kludge.
   First build a string containing an assignment, then get
   REXX to execute that string. We end up with a variable
   containing the contents of the Stem Item.
*/
zzName = 'zName = ' ID || '.Name'          /* build string 'zName =
                                           USERx.NAME' */
INTERPRET zzName                          /* Execute above string
SAY 'Name = ' zName                       /* zName contains value
                                           in USERx.NAME */

zzPass = 'zPass = ' ID || '.Password'
INTERPRET zzPass
SAY 'Password = ' zPass
zzCon = 'zCon = ' ID || '.ConnectDate'
INTERPRET zzCon
SAY 'Connected On = ' zCon
zzMach = 'zMach = ' ID || '.MachineType'
INTERPRET zzMach
SAY 'Machine Type = ' zMach
RETURN
```

Figure 14 - Using STEM variable as Structures

A subroutine can be called from any point in your program. The problem with this is that the variable names you use within your subroutine may be used for a different purpose somewhere else in your program. This can result in corruption and errors when you run the program. REXX has a special keyword, PROCEDURE. This tells REXX to create a barrier, the variables on one side of the barrier are distinct from those on the other side. This allows you to use two separate variables with the same name. Figures 12a and 12b show the two cases.

Subsystems

External subsystems are used in the same fashion as REXX subroutines. The only difference is that they must be registered first. This is done by call-

ing the RxFuncAdd function. This function registers the new function with REXX. The new function may then be used in the normal way. The OS/2 REXX provides a utility library (RexxUtil). This allows access to a wide range of OS/2 operating system functions from within a REXX program. Another example is the DB2/2 interface. This functions the same way. You register two new functions SQLEXEC and SQLDBS. These are then used to execute DB2/2 database commands against a local, or remote, database.

Stems

There is a lot you can accomplish with normal variables, but for any complex programming task it helps to have the ability to build complex data

structures. These normally fall into two categories, aggregate types and structured types. Aggregate types are collections of data where the data is all of the same type. For example, a table of timings for a benchmark. A structured type is where the data is related, but may be of varying types. An example of this might be the configuration details of a machine, where you have its manufacturer, model number, memory, disk drives, video card type, disk drive size, etc. These could be handled by using individual variables, but it is easier to have constructs that allows you to handle these as a single item.

In languages like C, or Pascal, these are built using arrays and structures or records. REXX does not have any of these constructs. What it does have, is something far more powerful, the STEM variable.

The STEM variable construct can be used to build arrays, or structures. Figure 13 shows a simple stem variable. It consists of a base variable name, followed by one, or more, sub-levels. Each sub-level is marked by a period. If the sub-level variable is a number, you have an array. If it is a name you have a structure.

One useful convention to follow (this is not mandatory but it helps) is to set the zero index (variable.0) to the number of active elements in the array. In our example we use RESULT.0. If we need to make the array bigger we can add the new elements, update RESULT.0 and then re-run the program. All sections of the code that need to access the size can use RESULT.0 and thus be portable.

In Figure 13 you can see the use of an index variable (X). This is used to access a value in the array - it is incremented to the next element, and the process is repeated.

Building Structures

The other use of stem variables is to build structures. Figure 14 shows an example. Here we are storing information about two users. This is captured in a stem variable. The items consist of Name, Password, ConnectDate and MachineType. The name of the stem is passed to a subroutine. This is where

```
/*
   Program to list date and time for all
   DEVICE= statements in Config.sys
*/

File = 'C:\CONFIG.SYS'           /* Name of File to Open */

/* Execute Loop till no more data */
DO UNTIL State = 'NOTREADY'
  aLine = LINEIN(File)           /* Read a Line */
  state = STREAM(File, 'S')      /* Set Status */
  CALL ProcessLine aLine
END
EXIT

/* This routine parses the line passed to it looking for DEVICE=
*/
ProcessLine:
PROCEDURE
ARG aLine

/*
   Use the PARSE UPPER command to look for '=' the word before
   is placed in CMD, the rest of the line in REST the UPPER
   keyword translate the line to uppercase
*/
PARSE UPPER VAR aLine cmd '=' rest
IF cmd = 'DEVICE' THEN
  DO
    /* parse again to remove any parameters to the device
       statement */
    PARSE UPPER VAR rest devfile ' ' rest

    /* Use the STREAM QUERY DATETIME function to get the
       creation date */
    fdate = STREAM(devfile, 'C', 'QUERY DATETIME')

    /* Print Date/Time and File Name */
    SAY fdate ' ' devfile
  END

/* Any non DEVICE = lines are ignored */
RETURN
```

Figure 15 - Reading a file using LINEIN

Understanding REXX

```

/*      Program to list date and time for all  DEVICE=
statements in Config.sys and to List all SET
Statements.  Output can be directed to the printer
if required. */
SAY 'Do you want to print the output (Y/N) ?'
PULL yesPrint
File = 'C:\CONFIG.SYS'          /* Name of File to Open */
Prt = 'PRN:'                    /* device name for Printer */
if yesPrint = 'Y' THEN STREAM(Prt, 'C', 'OPEN WRITE')
/* open printer for writing */
STREAM(File, 'C', 'OPEN READ')
/* Explicitly open file for read */
/* Execute Loop till no more data */
DO UNTIL State = 'NOTREADY'
    aLine = LINEIN(File)        /* Read a Line */
    state = STREAM(File, 'S') /* Set Status */
    CALL ProcessDevLine aLine
END
STREAM(File, 'C', 'SEEK 0')     /* reposition File at beginning */
DO UNTIL State = 'NOTREADY'
    aLine = LINEIN(File)        /* Read a Line */
    state = STREAM(File, 'S') /* Set Status */
    CALL ProcessSetLine aLine
END
STREAM(File, 'C', 'CLOSE')
if yesPrint THEN STREAM(Prt, 'C', 'CLOSE') /* Close Printer */
EXIT
/* This routine parses the line passed to it looking for
DEVICE= */
ProcessDevLine:
PROCEDURE
ARG aLine
PARSE UPPER VAR aLine cmd '=' rest
IF cmd = 'DEVICE' THEN
    DO
        . . . . . oLine = fdate ' ' devfile
        SAY oLine
        IF yesPrint = 'Y' then LINEOUT(Prt, oLine)
/* Output line to Printer */
    END
RETURN
ProcessSetLine:
. . . . . Code to process SET statements
RETURN

```

Figure 16 - Using explicit Open and Close

ADDRESS	- Set Default Command Destination
ARG	- Extract Passed parameters
CALL	- Call a subroutine or external REXX Routine
DO	- Start a code block or loop construct
DROP	- Discard a variable (unassign it)
EXIT	- Terminate a program
IF	- Test and Branch
INTERPRET	- Interpret a string as a REXX Command
ITERATE	- Terminate this count through a loop and move on to the next one (CONTINUE)
LEAVE	- Exit a Loop
NOP	- Do nothing
NUMERIC	- Set precision values
OPTIONS	- Pass requests to the default command processor
PARSE	- Scan and decode a string
PROCEDURE	- Start a procedure definition
PULL	- Get data from default input queue
PUSH	- Put data into the default input queue (LIFO)
QUEUE	- Manage a Queue
RETURN	- Return from a subroutine or function
SAY	- Print a message
SELECT	- Select from a number of test condition (multiway IF)
SIGNAL	- Jump to a label and continue execution from there (GOTO)

Figure 17 - Some useful OS/2 REXX Keyword Commands

we come across a major problem with REXX. It is very difficult to pass an entire stem to a subroutine. What we have to use is a kludge. First we build a string that contains an assignment for the stem plus tail. Eg, zName = USER1.NAME. This string is then executed (using INTERPRET). This places the current contents of the variable USER1.NAME into zName. We can then use zName in the normal way. Long winded, but it appears to be the only way in REXX to pass an entire stem into a subroutine.

The two types of usage can be combined, so we could design an array of structures, eg, USER.1.NAME, or we can build arrays with more than one dimension, eg, RESULTS.1.1. There is no limit, except those of the specific implementation we are using.

Storing Data

In a real world program you normally need to manipulate large amounts of data. You could of course enter all this data by hand, but the normal practice is to store it in a file. From its inception, REXX has been designed to work with files and text. Figure 15 shows a simple program. This reads the CONFIG.SYS file, searching for DEVICE= statements. When it finds one it displays the line, and the timestamp from the file. Notice that the program is broken down into two routines - the main routine reads each line and the ProcessLine routine is responsible for handling each line.

Streams

REXX, like many languages, treats files as one long sequence of characters. This sequence is called a STREAM. There are functions in REXX to handle this stream one character at a time, or as a complete line. Before you can use a file, you have to open it, and an open file can not normally be accessed by another OS/2 task.

In our main routine we use an UNTIL loop. This loop is terminated when there are no more lines to read in. To fetch each line we use the LINEIN function, which returns the next line from the file. Once we have read a

ABBREV	Returns True if String starts with an abbreviation
ABS	Returns Absolute (positive) value
ADDRESS	eturns name of current default environment
ARG	Returns information about an Argument
BEEP	Makes a Beep noise
BITAND	Bitwise AND
BITOR	Bitwise OR
BITXOR	Bitwise Exclusive OR
B2X	Binary to Hex Conversion
CENTRE/CENTER	Centre a string within another
CHARIN	Get character from a Stream
CHAROUT	Send characters to a stream
CHARS	Return count of characters in a stream
COMPARE	Compares two string
CONDITION	Return information about a trap condition
COPIES	Create a string of input string repeated a number of times
C2D	Character to Decimal Conversion
C2X	Character to Hex Conversion
DATATYPE	Determines the data type of input string
DATE	Get date
DELSTR	Delete a Substring
DELWORD	Delete a word from a string
DIGITS	Return current decimal precision
D2C	Decimal to String Conversion
D2X	Decimal to Hex Conversion
DIRECTORY	Returns name of current directory
ERRORTEXT	Get error message text
ENDLOCAL	Restores previous environment context
FILESPEC	Returns selected item from a file specification
FORM	Returns current setting of NUMERIC FORM
FUZZ	Returns current setting of NUMERIC FUZZ
INSERT	Insert a string into another
LASTPOS	Find the last position of a string in another
LEFT	Get left most characters in a string
LENGTH	Get length of a string
LINEIN	Read in a line from a stream
LINEOUT	Write a line to a stream
LINES	Number of lines in a stream
MAX	Maximum of a list of values
MIN	Minimum if a list of values
OVERLAY	Overlay a string on another
POS	Find the position on one string in another
QUEUED	Returns the number of lines in a Queue
RANDOM	Returns a random number
REVERSE	Reverse a string
RIGHT	Rightmost characters from a string
SETLOCAL	Start a local environment context
SIGN	Returns the sign of a number
SOURCELINE	Get the text for a specific line in the REXX program
SPACE	Pads a string with blanks
STREAM	Manage a stream
STRIP	Strip trailing and leading blanks from a string
SUBSTR	Return a substring from a string
SUBWORD	Return a specific string starting at a specific word
SYMBOL	Return if a word is a valid variable
TIME	Return current time
TRACE	Set tracing options
TRANSLATE	Convert a string based on a translate table
TRUNC	Return the integer part of a number
VALUE	Return the current value of a symbol
VERIFY	Verify a string contains another
WORD	Return a word from a string
WORDINDEX	Return the start position of a word in a string
WORDLENGTH	Return the length of a word in a string
WORDPOS	Searches a string for a word
WORDS	Number of words in a string
XRANGE	Returns a string containing a range of byte values
X2B	Hex to Binary Conversion
X2C	Hex to Character Conversion
X2D	Hex to Decimal Conversion

Figure 18 - OS/2 REXX Built-in Functions

line, we use the STREAM function to set a status flag. The line is passed to ProcessLine for further handling. In ProcessLine we use PARSE UPPER VAR with a template to look for separator strings, in our case an "equals" character. The input string (aLine) is searched for "=" - if it is found the contents of the string before the "=" are placed into CMD. The rest of the line is placed into REST. CMD is then checked for the word "DEVICE". The UPPER keyword ensures that the line is translated into upper case. This guarantees that we will find all instances of the word.

If the line does not contain "DEVICE" no further action is taken, otherwise the line is parsed again. This time we are looking for a blank. This is to ensure that if the device has any parameters, these are stripped from the file name. Finally, we use the QUERY DATETIME option of STREAM to get the creation date and time for the device driver. This information is then displayed.

In the above example we used the default behaviour of LINEIN. If a file has not been opened, then LINEIN will open it for you. Sometimes we may want to override this behaviour. Using STREAM we can explicitly open and close a file. We can also reposition the read or write position. Figure 16 shows a cut down version of figure 15 with explicit opens. Here, we display the DEVICE statements as before. Once these have been displayed we re-position the file to the beginning and re-read it. This time we process all the SET statements. We have also included the option of printing each line.



The Author

Geoff Drake is a consultant who specialises in OS/2, Object Oriented Technology and OS/2 Databases. When he is not doing that, he can be found fishing in a trout stream somewhere pleasant.

Assessing PC Users' Training Needs

In PCSA 64, Matthew Bell explained how to train PC users. In this article, Debi Schifreen explains in more detail the correct way to plan a training session.

Anyone who's ever dared to ask a professional trainer what it's like to work for just six hours a week in return for a full-timer's salary will probably be familiar with the standard response this question always produces.

The trainer will undoubtedly point to the vast amount of behind-the-scenes work necessary to ensure a smooth running training session and will argue that time spent actually in the training room accounts for only a tiny proportion of their workload.

The purpose of this article is to outline the activities which go on before and after a training session and which are necessary to ensure that the company maintains a strategy for staff training and development which will be of maximum benefit to all concerned.

Most of the advice in this article is equally applicable to both group training and individual training. However, since group training requires a greater amount of planning (as there are more things that can go wrong), most of the examples cited will concentrate on training people in groups rather than on a one-to-one basis.

Objectives

Managers all too often organise a training session without detailing exactly what they require their students to be taught. The risk they therefore take is that even if the session does run "successfully", ie, the participants learn everything they were supposed to learn, they will, at the close of the session, be no better at their job than they were at the beginning.

The primary task with regard to planning is to detail what each partici-

pant will need to know by the end of the session. To do this, it is often helpful to start by gathering some information from the students themselves or from their immediate superior regarding their job description. Ask the trainee what they would like to be taught. Ask them if they've experienced any problems recently which wouldn't have occurred had they had a better understanding of anything in particular. Listen to what people say and try to identify as many areas of common problems as possible. Then base the training on these points.

When a large number of people are to be trained, it may well only be feasible to spend a few minutes with each person to discuss their training needs. On the other hand, in cases where training is to be carried out on a one-to-one basis or with just two or three people, the process of identifying training needs can be greatly enhanced by following a more formal procedure called a job analysis. While not always necessary in simple training exercises, conducting a JA will always improve the quality and effectiveness of training people in more complex areas.

Critical Incidents

Job Analysis is the process of collecting and analysing information about each of the individual tasks which collectively form a person's job. One of the most effective methods of assessing training needs is a specific interviewing strategy known as the critical incident technique. Here, the job holder is asked to list situations which have occurred during the previous week or two at work and to categorise them into two sets - those which they particularly associated with a feeling of success and those which they linked with a sense of failure. Of course, it is important to reassure the member of staff that their responses will be strictly confidential. Typical examples might be:

Successes

- a) I cleared a backlog of data to be analyzed.
- b) I redesigned three standard letters and was congratulated on the improvements I had made.
- c) I dealt with a customer's complaint and prevented him from going to a competitor.

"Ask the trainee what they would like to be taught. Ask them if they've experienced any problems recently which wouldn't have occurred had they had a better understanding of anything in particular."

Failures

- a) I accidentally deleted 10 pages of data and had to spend two hours re-typing it.
- b) I was reprimanded for sending out a letter with three spelling mistakes.
- c) A customer complained about me because I promised him an answer to his query within 15 minutes and then failed to contact him until the following day.

In addition to the job holder him/herself, other employees can give valuable insight into areas in which training might be valuable for a colleague. Two of the best resources will be the individual's immediate superior, and a technical expert.

The jobholder's manager or supervisor will undoubtedly be willing (perhaps too willing) to identify areas of their own workload which could be carried out more effectively if other tasks could be delegated. They may for example cite a particular day when they had to put their own workload on hold in order to assist someone who was experiencing difficulties. Similarly, they may be spending a considerable amount of each day on a time-consuming task purely because no one else knows how to do it.

"Managers all too often organise a training session without detailing exactly what they require their students to be taught."

After gathering this type of information, if at all possible, and if appropriate, a technical expert could be called in. Such an individual could probably provide insight into whether or not problems expressed are the result of insufficient training. For example they can answer questions like "Processing an order into the system takes at least 30 minutes. How can people be taught to be quicker?"

Expressing Needs

Once the training needs have been identified, it is important to express them in a way so as to make them measurable and definable. It is too vague to say that trainees should have a thorough understanding of the order processing software. Break the task down into clear, concise fragments such as:

- Staff should be able to input an order form in five minutes.
- Once input, staff should be able to answer five questions on customer details within 20 seconds by accessing the relevant information on the screen.
- Staff should be able to edit information, eg, to change the spelling of a name or address on a given order.

Knowledge and Skills

Having assessed the training needs and outlined precisely what your group of trainees require help with, the next stage is to consider the form that the information should take in order for it to be of maximum benefit. Basically, training needs can fall into two categories:

- The need for additional factual knowledge.
- The need for additional practical skills.

For example, learning to drive a car requires knowledge of what pedals and switches to operate plus the practical skill of spatial ability and coordination. Teaching requires having access to the information you are going to teach plus some sort of skill at public speaking.

Knowledge without skills and vice versa is hopeless and there are, of course, countless equally applicable examples in the field of computers. The most user-friendly WordPerfect training package on the market will be of no use to someone who can't type.

The reason this distinction has implications for training preparation is to avoid time being wasted in training someone in a task which they are not really capable of doing. If someone is

9.30	General chatting and waiting for latecomers.
9.45	Distribute sets of instructions showing how to enter WordPerfect, create and save a text file. Practice individually.
10.00	Divide the group so as to have three participants sharing one PC. Distribute a 100-word hand-written article. Ask the member of each group with the fastest typing speed to type in the article.
10.15	Move away from the machines to face the front of the room. Using visual aids, give a talk on a selection of commands eg. delete, apell, upper case, centre, copy and print.
10.40	Coffee break
11.00	Instruct small groups to return to their article. Give a list of instructions eg. put into double spacing, underline the title, put all words of three letters into italics.
11.30	Return to individual desks. Issue task of entering WordPerfect, typing in personal name and address to a specified format, save on hard disk and on floppy disk, and print.
11.45	Questions and answers.

Figure 1 - Sample WordPerfect training schedule

Training Needs

"Once the training needs have been identified, it is important to express them in a way so as to make them measurable and definable."

experiencing problems at their work this may be because they require more practice rather than because there are things they do not know. It is important, therefore, when planning training to look at the proposed training needs and decide which are the areas in which people require help to develop their skills rather than just be given more information.

Scheduling

Once the training needs and training materials have been identified, it should finally be possible to start outlining a timetable for the session to follow. This is an essential part of planning as the format of a training session should never be taken for granted or left to chance. The best way of illustrating potential disasters is to use genuine past experiences. A personal one which springs to mind is the time I walked into a classroom to face a group of 15 students who had turned up for a two-and-a-half hour seminar. Equipped with an 18-page handout which I distributed to the group, I commenced the session. Unfortunately I'd made the handout too self-explanatory, which meant that no one felt the need to bring up any points for discussion. Also, I hadn't incorporated any active tasks for the group to carry out, meaning that the class quickly became lethargic and bored. The result was that I covered everything I'd intended to cover in 30 minutes, leaving two whole hours (which seemed more like two whole lifetimes) to ad lib.

There is only one foolproof way to totally eliminate this kind of disaster and that is meticulously to plan each step of the session. Decide on the methods you are going to use to put

the information across. Then plan a series of stages ensuring that each stage uses a different method and lasts for around 20 minutes.

Training Methods

There is more than one way of teaching and no single method is good enough on its own except in the case of a very quick and simple training session. In longer sessions people get bored easily and this leads to rapidly decreasing levels of concentration. To combat this there are two specific rules to bear in mind:

- 1 Training is most effective when the trainee is actively learning rather than passively listening.
- 2 Learning is greatly enhanced when people work together as a team.

Active vs Passive

Give a toddler 10 chocolates, then take six away and they will be quick to express their awareness that they have been given a raw deal. However, then try teaching the child that $10-6=4$ and their understanding of subtraction will rapidly increase.

Similarly, if you sit someone down in front of a computer and leave them alone to experiment they will pick up a considerable amount of information which they may not have acquired through listening to a lecture. Left alone to explore, the individual is like a sponge. It soaks up information from the environment without really having to try. On the other hand, when sitting passively in front of a blackboard, the mind is less like a sponge and more like a hard, shiny stone. Information can be thrown at it but, because the person is not free to absorb

it at their own pace, most of it will slide straight off. When planning a training session it is of great value to bear this in mind. Be sure to have an equal balance of active and passive learning.

Learning As A Team

The well worn cliché of two heads being better than one comes into dramatic reality in a learning situation. If you sat a novice computer user in front of a PC and told them they had one hour to work out how to type and print a letter in WordPerfect, then a large percentage of people of average intelligence would probably fail. If, on the other hand you put six trainees together and gave them the same task, chances are they would do it in half the time. The phenomenon which occurs is that people working as a team feed off each other's ideas. One member of the group will work out a solution to the first stage of the problem and will

- 1) Did you find the material
 - a) too technical?
 - b) too easy?
 - c) about right?
- 2) Was the training
 - a) boring and long-winded?
 - b) interesting and lively?
 - c) in between?
- 3) How much of the material was completely new to you?
 - a) none to 20%
 - b) 20% to 50%
 - c) 50% to 80%
 - d) 80% to 100%
- 4) Do you now feel more confident about your work?
 - a) very much so
 - b) slightly
 - c) not really
 - d) not at all

Figure 2 - Sample post-course questionnaire

"A secretary who has just been trained to use some new software will lose around a quarter of what they have been told if they have to spend the subsequent two days clearing a backlog of filing."

then get stuck. Subsequently, a second person who until that point had had no idea how to get started, will then be able to work out how to progress a little further having been given a start. It is in the trainer's interest to incorporate some group work in the training program.

Figure 1 shows an example of a training schedule for teaching a group of 12 secretarial staff the basics of WordPerfect.

Dealing With Mixed Ability

When devising a tutorial for a group of people, it's vital to bear in mind that each and every member of the group will be at a different level. The best strategy is to set the standard at a level of understanding slightly higher than that of the lowest ability members of the group. This should be ideal for those at the middle of the range. The more experienced members could then be used in the group situation to assist the less experienced ones.

Transferability

It is worth bearing in mind that any training is pointless if it is not transferable to the work situation. How many times have you heard someone say "I could do all the practice examples in the classroom but when we had to do it for real, my mind went blank"? In order to ensure that skills learned in training are taken back to the office two conditions are required:

- 1 The training must consist of real tasks taken from the workplace.
- 2 The time span between attending

training and returning to the office must be minimal. A secretary who has just been trained to use some new software will lose around a quarter of what they have been told if they have to spend the subsequent two days clearing a backlog of filing.

Evaluating Training

In most working environments, training courses should not be a one-off event. Every six months or so, the strengths and weaknesses of existing employees should be reassessed, allowing new areas for training to be identified. Similarly, typical rates of staff turnover will probably dictate a need for repeating standard training programmes on a fairly regular basis.

With this in mind, it is therefore necessary also to assess and modify the training courses themselves. Two areas in particular require some kind of explanation.

First, and of prime importance, is discovering whether the training course was effective. In other words, you need to discover the extent to which the trainees actually are carrying out their job more efficiently than they had been prior to training. To do this, go back to the initial set of training objectives and assess each trainee on each objective. This highlights the necessity for setting workable and measurable objectives in the first place.

The second type of evaluation is that which examines the trainee's view of their experience of the course. It goes without saying that the more the participants enjoy the training, the

more they will put into it. Finding out what they thought of the course, therefore, will have valuable implications for future training. A simple post-course questionnaire will be the best tool to achieve this, and an example is shown in Figure 2. At the end of training, ask everyone who took part if they would mind filling in a short form about their attitudes to the course. Keep this very short as, by the time the training has ended, few participants will have much concentration left.

Make all questions multiple-choice, and don't leave any space for free-text answers. This way, it's easier to analyse the results and, if you amend the course itself, find out whether the amendments have improved the end result.

PCSA

The Author

Debi Schifreen MSc is a freelance occupational psychologist specializing in staff assessment and training. She can be contacted via PCSA's London offices or on CompuServe as 100016,301.

Understanding PCMCIA

Dave Stott presents the second half of his description of PCMCIA technology, a major component of modern laptop machines.

In part one of this feature on PCMCIA technology I dealt with the PCMCIA specification and the technical details of the interface itself. I also covered the various memory card types and the suitable application of those different types currently available to address particular needs.

This concluding part covers the extremely wide variety of I/O cards available that can be used to greatly enhance the capabilities of a PCMCIA-compliant system. The majority of such PCMCIA I/O cards are designed to be direct equivalents of standard desktop PC expansion cards offering the same functionality in a much smaller, neater and more convenient package.

However, despite significant advances in miniaturisation, the physical size of PC Cards does place certain limitations on what can and can't be incorporated within the format. These limitations are most apparent with connectors, which are generally much more bulky than the cards themselves. The solution to this problem is usually solved with the use of a short "drop" or "interconnect" cable which is either an integral part of the card or is attached via a small multi-pin socket on the opposite edge to the PCMCIA interface itself.

This is the edge which protrudes from the PCMCIA socket and, as there is no formalised standard at present, PC Card manufacturers have adopted different ways of achieving this connection ranging from a 7 pin to a 15 pin socket. Fortunately, the PCMCIA has recognised this problem and they are about to ratify an extension to the PC Card standard for the connection of drop cables.

As Figure 3 shows, most PCMCIA I/O cards require Type II slots but some are available to fit Type I. At present all PCMCIA format hard disk cards need a Type III slot.

Communications Cards

Among the first I/O cards to be developed were communications cards such as the straightforward modem and the increasingly popular fax modem. These were developed in response to market demand as mobile users increased their desire to keep in touch via, for example, electronic mail or fax.

Although most portable PC manufacturers offered their own proprietary modems for mobile systems these tended to be very expensive and often quite difficult to obtain. Most modems were specific to a particular manufacturer's range and sometimes there would even be different versions for the various types of portable within a range. Naturally, as far as users were concerned, this was most unsatisfactory and the PCMCIA standard offered a much needed solution.

The first PCMCIA modem cards on the market were very basic data-only models with a maximum of either 1200 or 2400 bps. Gradually as the technology improved and demand increased, faster cards have become available.

Nowadays most manufacturers offer high specification fax modem PC Cards with a top speed of 14,400 bps, V42 error correction, V42bis MNP 5 data compression and Class III send and receive fax capabilities. Modem cards are now widely available and typically cost between \$300 and \$500 depending on specification. When used with software that has built-in fax capabilities, such as the latest versions of Word and WordPerfect, a laptop PC really can become a mobile office.

Today, apart from basic memory cards, the PCMCIA fax modem card is by far the most popular PC Card purchased by mobile computer users.

Wireless Communications

The need to use land-based telephone lines is the main factor restricting the mobility of users with portable systems equipped with standard fax modem cards. So it is not surprising that manufacturers are beginning to make use of the cellular radio network to provide less restrictive data communications for the mobile user.

At present there are several PC Cards which make use of wireless communications either connecting to a cellular phone or by using independent radio transceivers.

Megahertz Corporation has two versions of its Cellular fax modem card. The CC4144MIT is compatible with Mitsubishi 3500, 4000 and Diamondtel 20, 22 cellular phones, whilst the CC4144NEC is compatible with the NEC P100 series. These cards also come equipped with the ability to hook into a standard telephone jack for using land-based lines. Typical price for these cards is \$400.

Another wireless communication device is the Socket PageCard from Socket Communications Inc. This is a radio pager which can be used as both a standalone device or plugged into a PCMCIA type II slot. When used with a portable PC the PageCard can be accessed as a COM port for sending and receiving alphanumeric messages via a paging service. Windows-based software is supplied, giving full access to all the PageCard's standalone functions. At present this card will only work on US-based paging services and costs around \$350.

On the slightly more exotic front is the NavCard GPS (Global Positioning System) Receiver from Rockwell International. This PCMCIA card makes use of the GPS series of satellites, maintained by the US Department of

Defense, which can provide world-wide positioning and navigational data. This system is extremely accurate and can pinpoint your position to within 150 feet anywhere in the world, and at the same time determine your direction of travel and velocity. Although it is not cheap at around \$1,000, GPS hardware and software can be a very valuable tool for archaeological, geological and geophysical surveying etc.

Packet Switching

The potential for growth in wireless data communications is enormous and the market is set to explode over the next few years. However, rather than make use of the existing circuit-switched cellular network, primarily designed for voice traffic, manufacturers are increasingly turning to wireless packet-switched networks as a better solution for data traffic. Mobitex is a public, packet-switched, wireless data network which is available in the US, Canada, the UK, Sweden, Finland and Norway. Mobitex networks are also planned for Australia and the Netherlands, with continuing efforts to assist national operators to launch the service in the rest of Europe, Latin America and the Pacific Rim.

Mobitex radio modems will shortly be available in PCMCIA format. At first they are likely to be rather expensive but as the market expands and production volumes increase, the price of these modems will continue to fall.

Network Adapters

For the mobile user who needs to connect to a LAN when returning to base there are several PCMCIA network adapter cards available. Both Ethernet and Token Ring based networks are catered for.

The EN2212-2 EtherPair-PCMCIA from Accton Technology Corporation provides connection to 10base-T wiring via a media coupler. The more versatile EN2212-1 EtherDuo features a media coupler equipped for auto selection of 10base-T or 10base-2 so it can attach to either twisted pair or thin

Ethernet cabling systems. Both these cards have a 16 KB RAM buffer and will work with industry standard NE2000 drivers.

Socket Communications Inc. offer similar products which are also NE2000 compatible but have 64 KB RAM buffers for improved performance. A typical single media card costs about \$300 whilst dual media cards are around \$400.

For Token Ring networks there is the Avanti! AV-16/4 PCMCIA card from PPCP Limited in the UK. This card can run at either 4 Mbits/second or 16 Mbits/second and is compliant with the IEEE 802.5 Token Ring Standard. It supports popular network software, such as NetWare, LAN Manager, LAN Server, Banyan Vines and PC/TCP and comes complete with NDIS and ODI drivers. The card can support up to 254 link stations and allows Early Token Release (ETR). Price for this card is around \$600.

Interface Cards

Occasionally mobile users need to enhance the interface capabilities of their system and there are several PC Cards which enable them to do just this.

The most basic is the Serial Interface card which provides an additional serial port for the connection of printers, plotters, external modems, scanners etc. Once again Socket Communica-

tions Inc supplies such a card based on the industry standard 16550 UART chip for fast serial communications. The card features zero wait state operation and can be configured as COM 1, 2, 3 or 4. It comes complete with a permanently attached rugged drop cable with a Data-bus bit-9 connector. These are relatively cheap cards and you should be able to obtain one for about \$200.

Also available from a number of suppliers are SCSI adapter PC Cards. These are ideal for connecting things like external CD-ROM drive, Tape Streamers or large hard disks to portable PCMCIA equipped systems.

New Media Corporation manufacture both standard SCSI and SCSI-2 versions of these cards. The standard SCSI version has an average 500 KB/sec transfer rate and is suitable for medium speed, low power SCSI peripherals. The SCSI-2 version has an average 1 MB/sec transfer rate and is designed for use with high speed peripherals. Both versions can be supplied with either Centronics, DB25 or SCSI-2 type cable connectors. The New Media ASPI Manager supports true hot insertion and removal of the card while the host system is switched on.

Adaptec Inc offers a SCSI-2 product, the APA-460 SlimSCSI host adapter card. With its 2MB/sec transfer rate, the SlimSCSI is as fast as many conventional 16-bit desktop adapters.

PCMCIA card	Slot Required		
	Type I	Type II	Type III
Memory	✓	✓	
Fax modem		✓	
Wireless comms.		✓	
LAN adapters		✓	
Serial interface		✓	
SCSI interface	✓	✓	
Terminal adapters		✓	
Hard Disks			✓
Sound cards	✓	✓	
Video Capture		✓	
Multi Function cards		✓	

Figure 3 - Slots Required by PC Cards

PCMCIA

A reversible connector on the three foot detachable cable allows you to route the cable to either side of your computer. The peripheral end has a standard 50 pin SCSI connector.

One of the main advantages of the SCSI interface is its ability to support several peripherals and all of these cards can support up to seven daisy chained devices. SCSI PC Cards typically cost between \$300 to \$360.

Terminal Adapters

Users who wish to access IBM host systems from their portable PCs may need either terminal adapter cards, for local connection, or synchronous modems for remote access via dial up lines. Both types are available in PCMCIA card format.

Capstone's 3270 & 5250 Microlink Credit Card adapter enables connec-

tion to either 3270 mainframe or 5250 mid range IBM communications networks. The card itself is common to both systems and uses an intelligent Network Media Module drop cable. The Microlink adapter auto-configures for either coax or twinaxial cable attachment according to the Network Media Module connected and the software configuration selected.

In its 3270 configuration the adapter provides direct connection to IBM 3x74 control units and gives functional compatibility with the most popular 3270 terminal emulation software. The 5250 version provides direct twinaxial connection to IBM S/36, S/38 and AS/400 systems and is compatible with terminal emulation software written to support IBM's Enhanced 5250 adapter. These Capstone adapters are available for approximately \$700 each.

PPCP Ltd offer 3270 and 5250 Remote Dial up cards. The Avanti! 3270 Autosync is a V.22bis or V.32bis fax modem with automated dial up and logon. It emulates either a 3274 or 3276 cluster controller giving up to eight host sessions, five of which may be prime sessions. It has straight forward keyboard remapping and removes the need for expensive SDLC adapters.

The Avanti! 5250 Autosync provides the same fax modem facilities and emulates 5251 model 12 or 5294 cluster controllers. It can maintain up to nine simultaneous host sessions, three of which can be printer sessions. It is fully compatible with IBM PC Support for AS/400 and System 36 or 38. These Avanti! Remote Dial up cards are rather expensive at around \$1,100 each.

Hard Disks

Over the past few years some major advances in hard disk technology have enabled manufacturers to produce very small hard drives. Following on from the 3.5 inch form factor the 2.5 inch drive has become commonplace for use as the primary drive in most notebook computers. The latest drives use a 1.8 inch form factor as pioneered by Hewlett Packard. This 1.8 inch form factor has now been adopted for use in

Socket Communications Inc.

2501 Technology Drive
Hayward
CA 94545
USA
Tel: 1-510-670-0300
Fax: 1-510-670-0333

Rockwell International

4311 Jamboree Road
Newport Beach
CA 92660-3095
USA
Tel: 1-714-833-4655
Fax: 1-714-833-6898

Accton Technology Corporation

1962 Zanker Road
San Jose
CA 95112
USA
Tel: 1-408-452-8900
Fax: 1-408-452-8988

PPCP Ltd

Caledonian House
98 The Centre
Feltham
Middlesex
TW13 4BH
England, UK
Tel: (44)-81-893-2277
Fax: (44)-81-893-1182

I/O Magic

Freeway Technology Park
199 Technology Drive
Building #140
Irvine
CA 92718
USA
Tel: 1-714-727-7466
Fax: 1-714-727-7467

New Media Corporation

Irvine Spectrum
15375 Barranca
Building B101
Irvine
CA 92718
USA
Tel: 1-714-453-0100
Fax: 1-714-453-0114

Calluna Technology Limited

One Blackwood Road
Eastfield
Glenrothes
Fife
KY7 4NP
Scotland
UK
Tel: (44)-592-630810
Fax: (44)-592-630920

Adaptec Inc

691 South Milpitas Boulevard
Milpitas
CA 95035
USA
Tel: 1-818-365-6264
Fax: 1-408-957-7150

Capstone Technology

47354 Fremont Boulevard
Fremont
CA 94538
USA
Tel: 1-510-438-3500
Fax: 1-510-651-7035

Megahertz Corporation

USA
Tel: 1-801-272-6000
Fax: 1-801-272-6077

Figure 4 - Contacts

PCMCIA hard disk cards and most hard disk manufacturers already produce a few of these.

As with previous hard disk formats, initial capacities are fairly low with 60 or 80 MB being common. This resulted in a rather high cost per megabyte of storage. However, larger capacity hard disk PC Cards are starting to appear to meet demand. All the major hard disk manufacturers are striving to produce cheaper, higher capacity PC Card drives and products are available from Seagate, Conner and Ministor to name but a few.

As an example, Calluna Technology offer an 80 MB capacity hard disk at around \$565 and a 105 MB version at around \$659. They've also announced a 128 MB capacity version which is expected to sell for approximately \$699 giving a cost per megabyte of \$5.46. Whilst this is still more expensive than fixed hard disks the ability to easily remove and swap drives is a significant advantage. These are ideal for supplementary storage, transferring large amounts of information and can provide a very quick and efficient means of backing up the data on a portable.

Multimedia

The designers of portable computers are constantly attempting to match the functionality and capability of desktop equivalents, but the advent of multimedia has posed something of a problem. How can the same functionality be achieved in a portable system when, for example, CD-ROM requires a 5.25 inch form factor? Panasonic has just launched a Notebook PC, the CF-V21P, which features the latest 3.5 inch format CD-ROM drive. This model also has a PCMCIA slot which can accept either one Type III card or two Type II cards and, therefore, it can make use of PC Cards designed to provide further multimedia functions.

One such PC Card is the WAVjammer from New Media Corporation which is a 16-bit stereo sound card. A 6 inch drop cable connects the card to a Media Access Module which contains four 1/8" jacks for Stereo Line In and Line out, stereo microphone and stereo speakers. It is a Type I card

which is SoundBlaster, Adlib and Microsoft Sound System compatible. A New Media Sound driver is supplied for both DOS and Windows.

Also available from I/O Magic is the Tempo Audio card which is Adlib and Cardinal sound board compatible. In addition to Stereo Line In, Line Out and Microphone sockets it has a MIDI out interface. This is a Type II card. You can expect to pay around \$350 for a PCMCIA format sound card.

Another Multi-Media product from I/O Magic is the Focus Video card. This provides notebook computer users with the ability to capture motion video, screen stills, and live video overlays on their LCD screens. It is fully compatible with Video for Windows and accepts input from camcorders, VCRs, videodisc, or any standard NTSC or PAL compatible sources. Performance is directly relevant to the CPU power available and typically you could expect to achieve 15-24 frames per second at 160 x 120 pixel resolution. Expect to pay around \$600 for this card.

Multi-Function Cards

As with the earliest PCs when free expansion slots were often at a premium, portable systems generally only have a maximum of two PCMCIA slots available. This severely reduces their expansion capabilities and frequently users will require more functions than cards and available slots can provide. This problem was solved effectively on desktop PCs with the introduction of multi-function expansion cards and now the same approach is being used with the PCMCIA interface.

There are already a few dual or multi-function PCMCIA cards available. One such card is the Trio from Socket Communications Inc. This is a combined fax modem and Ethernet LAN adapter which features two separate drop cables, one for each specific connection. On the LAN interface both 10baseT and 10base2 media connections are provided.

The amazing thing about this card is that you can use the fax modem and the LAN adapter at the same time, so you can fax or email documents that

are stored anywhere on your network. Considering the combined capabilities offered by the Trio it is excellent value at around \$700 for a multi-function PC Card which only uses a single Type II PCMCIA slot.

Conclusion

The users of mobile computers are becoming more sophisticated and discerning in terms of their requirements. Nowadays they are unwilling to make compromises, often requiring comparable functionality to their desktop systems when out in the field. This is despite the fact that desktop systems are themselves becoming more and more flexible and powerful.

Fortunately, manufacturers worldwide have recognised these requirements and are using the flexibility and capabilities of the PCMCIA interface to produce a variety of solutions to address the demands of mobile users. The use of PCMCIA technology has been so successful that manufacturers can now even provide a high degree of Multi-Media functionality on portable systems.

As more and more PCMCIA equipped systems become available the demand for many different types of PC Cards will increase. Eventually demand will eliminate the price premium which currently applies to such products. As can be seen from the examples above there is already a wide variety of different PC Cards available offering a number of different functions and this trend is set to continue. All this is good news for the mobile user as PCMCIA looks set to become a significant factor which will contribute to the growth of portable computing in general.

PCSA

The Author

Dave Stott has worked in the computer industry for nearly 25 years. He is a freelance writer and IT consultant.